Fraunhofer
USA
Center for Experimental Software Engineering

The Evolution of Empiricism in Software Engineering: A Personal Perspective

University of Maryland
Fraunhofer Center for Empirical Software Engineering

basili@cs.umd.edu

Fraunhofer USA
Center for Experimental Software Engineering

Evolving the Software Engineering Discipline Model Building, Experimenting, and Learning

Outline

Evolving Software Engineering knowledge through experimentation

Nature of the Software Engineering Discipline

Evolution of the Discipline: What we have learned over 40 + years

Where do we have to go?

Fraunhofer USA

Center for Experimental Software Engineering

Evolving Knowledge
Model Building, Experimenting, and Learning

Understanding a discipline involves **building models**,
e.g., application domain, problem solving processes

Checking our understanding is correct involves

- testing our models
- **experimenting**

Analyzing the results of the experiment involves **learning**

We **encapsulate knowledge** into models**, test that knowledge,** and **evolve our knowledge** over time

The understanding of a discipline **evolves** over time

This is the paradigm that has been used in many fields,
e.g., physics, medicine, manufacturing.

Evolving Knowledge
Model Building, Experimenting, and Learning

What do these fields have in common?

They evolved as disciplines when they began applying the cycle of model building, experimenting, and learning

Began with observation and the recording of what was observed

Evolved to manipulating the variables and studying the effects of change in the variables

What are the differences of these fields?

Differences are in the **objects** they study, the **properties** of those objects and the system that contain them, the **relationship** of the object to the system, and the **culture** of the discipline

This effects

how the models are built

how the experimentation gets done

Evolving Knowledge
Model Building, Experimenting, and Learning

Physics

- understand and predict the behavior of the physical universe
- involves theorists and experimentalists
- has progressed because of the interplay between the two

Theorists build models to explain the universe

- predict the results of events that can be measured
- models based on
  theory about the essential variables and their interaction
  and data from prior experiments

Experimentalists observe, measure, experiment to

- test or disprove a hypothesis or theory
- identify new variables

But at whatever point the cycle is entered there is a modeling, experimenting, learning and remodeling pattern

Early experimentalists only observed, did not manipulate the objects
Modern physicists have learned to manipulate the physical universe, e.g. particle physicists.

Evolving Knowledge
Model Building, Experimenting, and Learning

Fraunhofer USA
Center for Experimental Software Engineering

Medicine
- aims at understanding the workings of the human body and the effects of various procedures and drugs on it
- There is feedback from practitioner to researcher to pratitioner

Researcher creates potential solutions and tests them in a variety of ways

Practitioner applies knowledge to cure and improve the human body and identifies new problems

Medicine began as an art form
- evolved as a field when it began observation and model building

Experimentation
- from controlled experiments to case studies
- human variance causes problems in interpreting results
- data may be hard to acquire

However, our knowledge of the human body has evolved over time

Evolving Knowledge
Model Building, Experimenting, and Learning

Fraunhofer USA
Center for Experimental Software Engineering

Manufacturing
- produce a product to meet a set of specifications
- understand the relationship between the process and the product

Manufacturing evolved as a discipline when it began process improvement

Relationship between process and product characteristics well understood

Process improvement based upon models of
- problem domain and solution space
- evolutionary paradigm of model building, experimenting, and learning
  e.g. Plan Do Check Act

Models are built with good predictive capabilities
- same product generated, over and over, same processes
- understanding of relationship between process and product

Software Engineering
The Nature of the Discipline

Like other disciplines, software engineering requires the cycle of model building, experimentation, and learning

The study of software engineering is a laboratory science

The researcher's role is to understand the nature of the processes, products and their relationship in the context of the system

The practitioner's role is to build "improved" systems, using the knowledge available

More than the other disciplines these roles are symbiotic

The researcher needs laboratories to observe and manipulate the variables
- they only exist where practitioners build software systems

The practitioner needs to better understand how to build better systems
- the researcher can provide models to help

Software Engineering
The Nature of the Discipline

Fraunhofer USA
Center for Experimental Software Engineering

Software engineering is development not production

The technologies of the discipline are human based

All software is not the same
- there are a large number of variables that cause differences
- their effects need to be understood

Currently,
- insufficient set of models that allow us to reason about the discipline
- lack of recognition of the limits of technologies for certain contexts
- there is insufficient analysis and experimentation

Evolution of the ESE Discipline

Fraunhofer USA
Center for Experimental Software Engineering

- Phase I (~1970s)
  - Isolated studies
- Phase II (~1980s)
  - Multiple Studies in one domain
- Phase III (~1990s)
  - Tying Studies Together
- Phase IV (~2000s)
  - Expanding Studies Across Domains and Environments
- Now and the future

Fraunhofer USA
Center for Experimental Software Engineering

Phase I: ~ 1970s
Running isolated studies for a particular purpose

Kinds of Questions:

Can we quantitatively measure the effect of the application of a particular approach on the product?

Can we distinguish products developed with different methods and techniques?

Fraunhofer USA
Center for Experimental Software Engineering

Phase I
Running isolated studies for a particular purpose

- **Situation at the time:**
  - Minimum amount of guidance for experimentation in the Software Engineering domain
  - Little or no useful data being collected
  - Models mostly based on theory rather than data
  - Modeling mostly restricted to cost estimation
  - Minimum opportunities for publication
  - Hard to convince the community this was important
  - …

Fraunhofer USA
Center for Experimental Software Engineering

Phase I: Personal Example
Iterative Enhancement Case Study

- **Problem**: Can we quantitatively measure the effects of the application of an incremental development method on the product over time?

- **Iterative Enhancement Method**:
  - produced incremental versions of the product,
  - each with more functionality and
  - used feedback from the previous version to improve the architecture, design, functionality

- **Experiment**:
  - Made quantitative observations over time, for each version
  - Measured changes in the architecture, design, structure and functionality comparing incremental products
  - Able to show improvements in the chosen measures

Phase I: Personal Example Methodology Evaluation Controlled Experiment

- **Problem**: Can we measure and differentiate the effects of different processes and teams on products?

- **Methods**:
  - Teams of three using structured programming and chief programmer teams (A)
  - Teams of three using structure programming (B)
  - Single Programmer (C)

- **Experiment**: A controlled experiment replicated study with three treatments
  - Able to show product A looked more like product C
  - Product A did better than products Band C with respect to proxy's for defects and effort

Phase I
Running isolated studies for a particular purpose

- **Learned:**
  - About running controlled experiments
  - Running an empirical study
  - Using nonparametric statistics
  - About developing automated metrics (proxies)
  - Limits of automated measures
  - Evaluation needs a basis for comparison (baselines)
  - A measureable relationship between process and product
  - Can quantitatively measure the effect of the application of one particular approach on the product
  - Products developed with different methods and techniques can show different product characteristics

Fraunhofer USA
Center for Experimental Software Engineering

Phase II: ~1980s
Multiple Studies in one Environment & Domain

Kinds of Questions:

Can we use other organization's models as they are?

Can we build baselines of various project variables (defects, effort) and identify where methods might make a difference?

Can we collect, analyze and store large amounts of project data and feedback information to the project and the organization?

Can we increase the quality of a particular class of systems over time by observing and identifying opportunities for improvement?

Phase II
Multiple Studies in one Environment & Domain

- **Situation at the time:**
  - Can measure various attributes about projects and products
  - Can learn through measurement
  - A small community began to take such studies seriously
  - Tutorials on measurement were available at conferences
  - Had an opportunity to do longitudinal studies
  - …

Phase II: Personal Example
Software Engineering Laboratory (SEL)
Observation, Feedback, Learning, Packaging

Fraunhofer USA
Center for Experimental Software Engineering

Consortium of NASA/GSFC, CSC, UMD (1976 – 2001)

Goals were to

- better understand software development
- improve the process and product quality for Ground Support Software

Used the NASA software developments as a laboratory to observe, build models, test hypotheses, package and evolve,

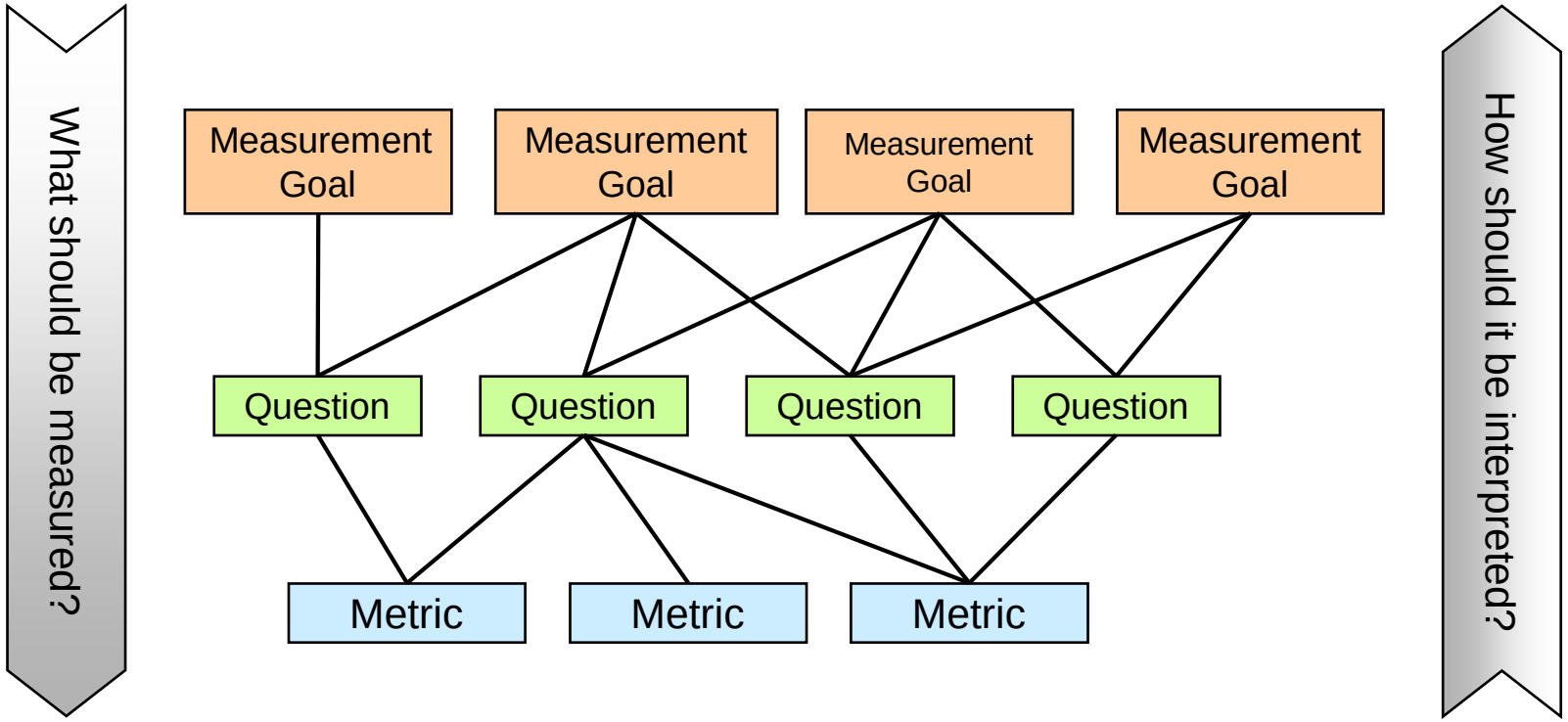
Kept the business going with an aim at improvement, learning, i.e., work was done on live systems

Used the University to test high risk ideas off-line

Learned what worked and didn't work, applied ideas when applicable

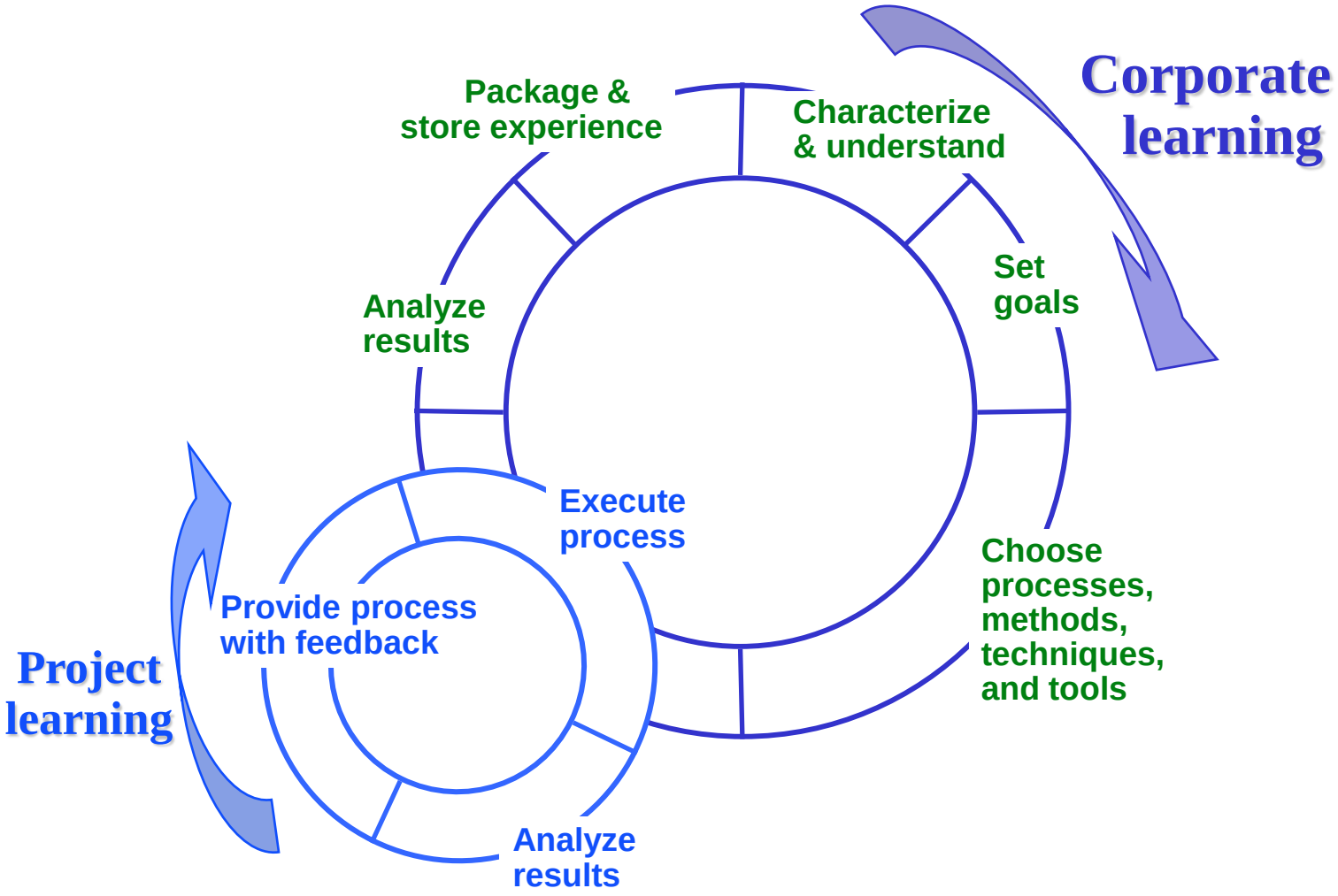
Developed new technologies, methods and theories when necessary

The GQM Structure

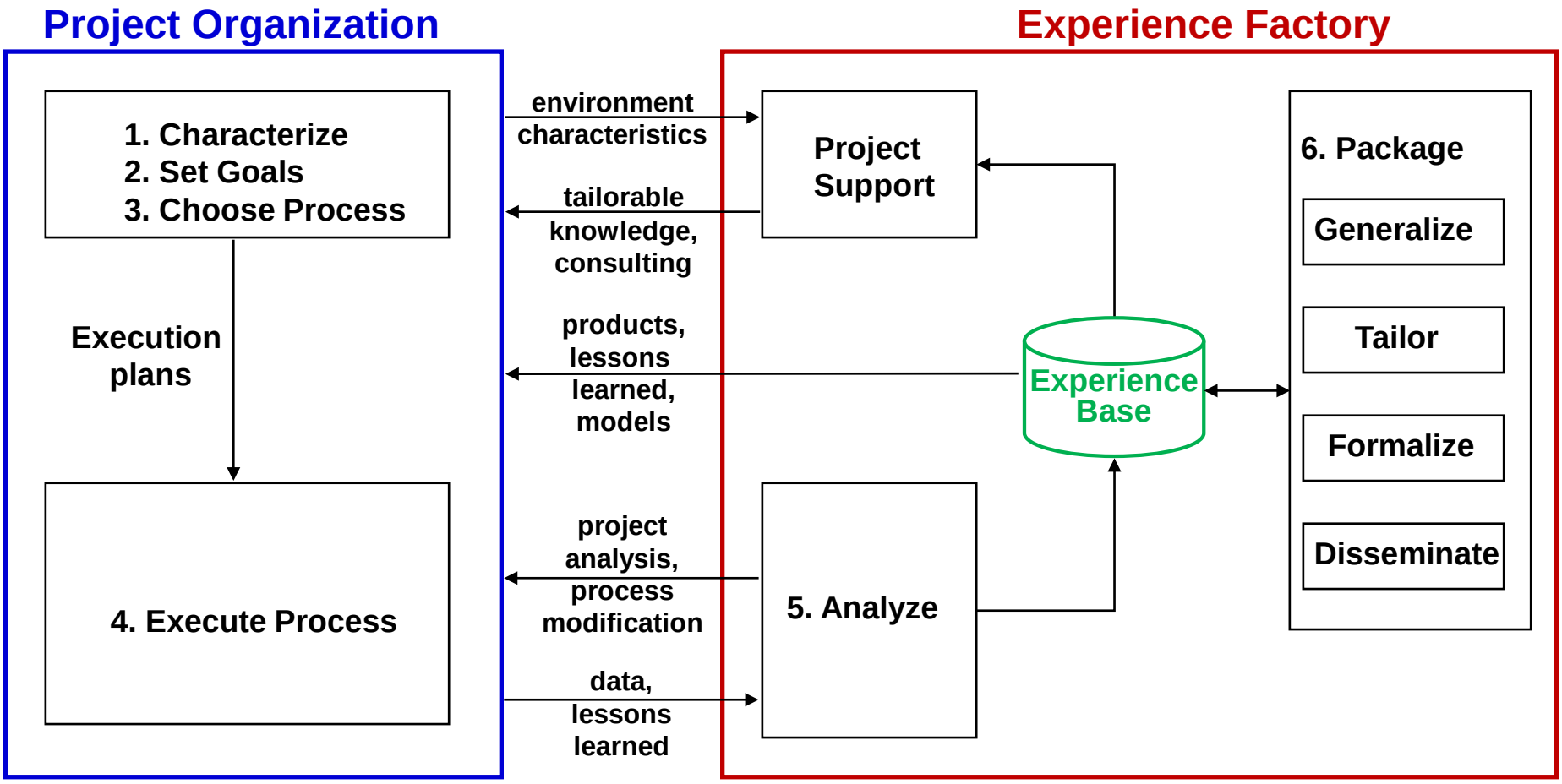


GQM: A mechanism for defining and interpreting operational, measurable goals

Quality Improvement Paradigm (PDCA for the software domain)



Organizational Structure for Learning



Continuous Improvement in the SEL

Fraunhofer USA
Center for Experimental Software Engineering

Continuous Gains

Decreased **Development Defect rates** by
75% (87 - 91) **37%**(91 - 95)

Reduced **Cost** by
55% (87 - 91) **42%** (91 - 95)

Improved **Reuse** by
300% (87 - 91) **8%** (91 - 95)

Increased **Functionality** five-fold (76 - 92)

With a cost of about 10% of development costs

The SEL was the winner of the first IEEE Computer Society Award for Software Process Achievement in 1996 and **CSC** was certified as CMM level 5

Phase II
Multiple Studies in one Environment & Domain

- **Learned:**
  - Data collection needs to be goal driven
  - About applying the scientific method to the software domain
  - Importance of understanding the environment (context)
  - Can build models that really represent the environment
    - Rather than general models
  - A variety of experiences can be reused, e.g., process, product, resource, defect and quality models within the environment
  - All experience needs to be evaluated, tailored, packaged for reuse in a variety of ways, and integrated

Fraunhofer
USA
Center for Experimental Software Engineering

Phase III ~1990s
Tying studies together

Kinds of Questions:

Can we *reduce the risk* in trying out a new approach?

Can we apply a *mix* of controlled experiments, case studies, quasi-experiments, qualitative analysis, and simple observations tied together to support hypotheses, reduce the threats to validity, and better understand the effect of context?

Are some techniques, methods more effective than others for different product characteristics, e.g., types of defects?

Phase III
Tying studies together

- **Situation**:
  - Context variables were beginning to be recognized as critical
  - People took notice of the SEL
  - A community was emerging:
    - International Software Engineering Research Network (ISERN) (1993)
    - Journal of Empirical Software Engineering (EMSE) (1996)
  - …

Running Multiple Types of Experiments

		#Projects	
		One	More than one
# of Teams per Project	One	Single Project (Case Study)	Multi-Project Variation
	More than one	Replicated Project	Blocked Subject-Project

Series of Studies

Code reading, functional and structural testing
Unit test size programs seeded with faults
Blocked subject-project: Fractional factorial design
Three replications: 42 UM (2), 32 NASA/CSC
Different strengths for different defect types

# of Teams	One		
per Project	More than one		**Reading vs. Testing**

Series of Studies

Scaled up to teams and larger projects
Compared 15 teams using Cleanroom and not using it
When reading is motivated it is very effective

# of Teams per Project	One		
	More than one	2. Cleanroom at Maryland	1. Reading vs. Testing

Series of Studies

Training and tailoring of Cleanroom for the SEL
Integrated into the existing process
Very effective on live project (40K SLOC) at NASA

		ne	More than one
# of Teams	One	**3. Cleanroom (SEL Project 1)**	
per Project	More than one	**2. Cleanroom at Maryland**	**1. Reading vs. Testing**

Experimental Learning Mechanisms

Effective over a series of projects
Some modification for contracted out projects
Recognized need to use reading at higher level, e.g. Requirements reading

		One	More than one
# of Teams per Project	One	**3. Cleanroom (SEL Project 1)**	**4. Cleanroom (SEL Projects, 2,3,4,...)**
	More than one	**2. Cleanroom at Maryland**	**1. Reading vs. Testing**

Experimental Learning Mechanisms

Fraunhofer USA
Center for Experimental Software Engineering

Series of Studies

Developed perspective based reading techniques
Experimented with requirements reading
Effective on controlled experiments with NASA developers

# of Teams	One	**3. Cleanroom (SEL Project 1)**	**…room …ojects, 2,3,4,...)**
per Project	More than one	**2. Cleanroom at Maryland**	**1. Reading vs. Testing** **5. Scenario reading vs. ...**

Phase III
Tying studies together

Fraunhofer USA
Center for Experimental Software Engineering

- **Lessons Learned:**
  - Can evolve a process by learning from multiple approaches
  - Can build confidence in a theory based upon multiple treatments
  - Can reduce risk by running smaller experiments off-line
  - Different techniques, methods may be more effective for different types of defects (process/product relationship)
  - Techniques can be developed based upon specific goals
  - Reading techniques can be developed and are valuable
  - Reading needs to be motivated
  - Software is more prevalent in organizations than top executives realize…

What happened to the SEL?

- The SEL activities lasted 25 years (1976-2001), when there was a total reorganization at NASA/GSFC
- The manager two levels up, who had been a strong supporter of the SEL, retired
- SEL activities had only been applied to ground support systems which were now contracted out based upon our success, so there was no home for the SEL
- The experience base was limited to GSS and lasted for several more years, available to anyone who wanted to use it
- After about 5 years GSFC realized they needed to improve their processes and have tried other approaches, CMMI

Phase IV ~2000

Expanding across domains, environments, technologies

Kinds of Questions:

Can we build a body of knowledge about a *technology* supported by empirical evidence?

Can we build a body of knowledge about a *domain* supported by empirical evidence, as with Ground support systems?

Can we build a *decision framework* to evaluate and choose among software development technologies?

Can we create an empirical *research engine* to provide the empirical evidence of what works and when?

Phase IV

Expanding out across domains, environments, technologies

Situation at the time:

Examples of studies building knowledge about a domain

Conferences and Journals opened up to empirical research

The community of researchers continued to expand

Various kinds of replications were being published

A rich palate of experimental methods were in use

Context variables began being studied and characterized and we began building knowledge across different contexts

Phase IV: Personal Examples

Expanding out across domains, environments, technologies

NSF Center for Empirically Based Software Engineering

Enable a decision framework, experience base, and empirical research engine to provide evidence of what works and when

Partners: Victor Basili (UMD), Barry Boehm (USC)

NASA High Dependability Computing Program

Elicit the software dependability needs of various stakeholders and the technologies to achieve that level of dependability

Partners: NASA, CMU, MIT, UMD, USC, UW, FC-MD

DARPA High Productivity Computing Systems

Build sufficient knowledge about the high end computing (HEC) so you can improve the time and cost of developing these codes

Partners: Lincoln Labs, MIT, UCSD, UCSB, UMD, USC, FC-MD

Fraunhofer USA

Center for Experimental Software Engineering

Phase IV: Personal Example
CeBASE: Center for Empirically Based Software Engineering

CeBASE Project Goal: Enable a **decision framework and experience base** that forms a basis and infrastructure needed to evaluate and choose among software development technologies

CeBASE Research Goal: Create and evolve an **empirical research engine** for building the research methods that can provide the empirical evidence of what works and when

Partners: Victor Basili (UMD), Barry Boehm (USC)

CeBASE NSF Center for Empirically Based Software Engineering

University of Maryland • Fraunhofer Center-MD • University of Southern California • University of Nebraska • Mississippi State University

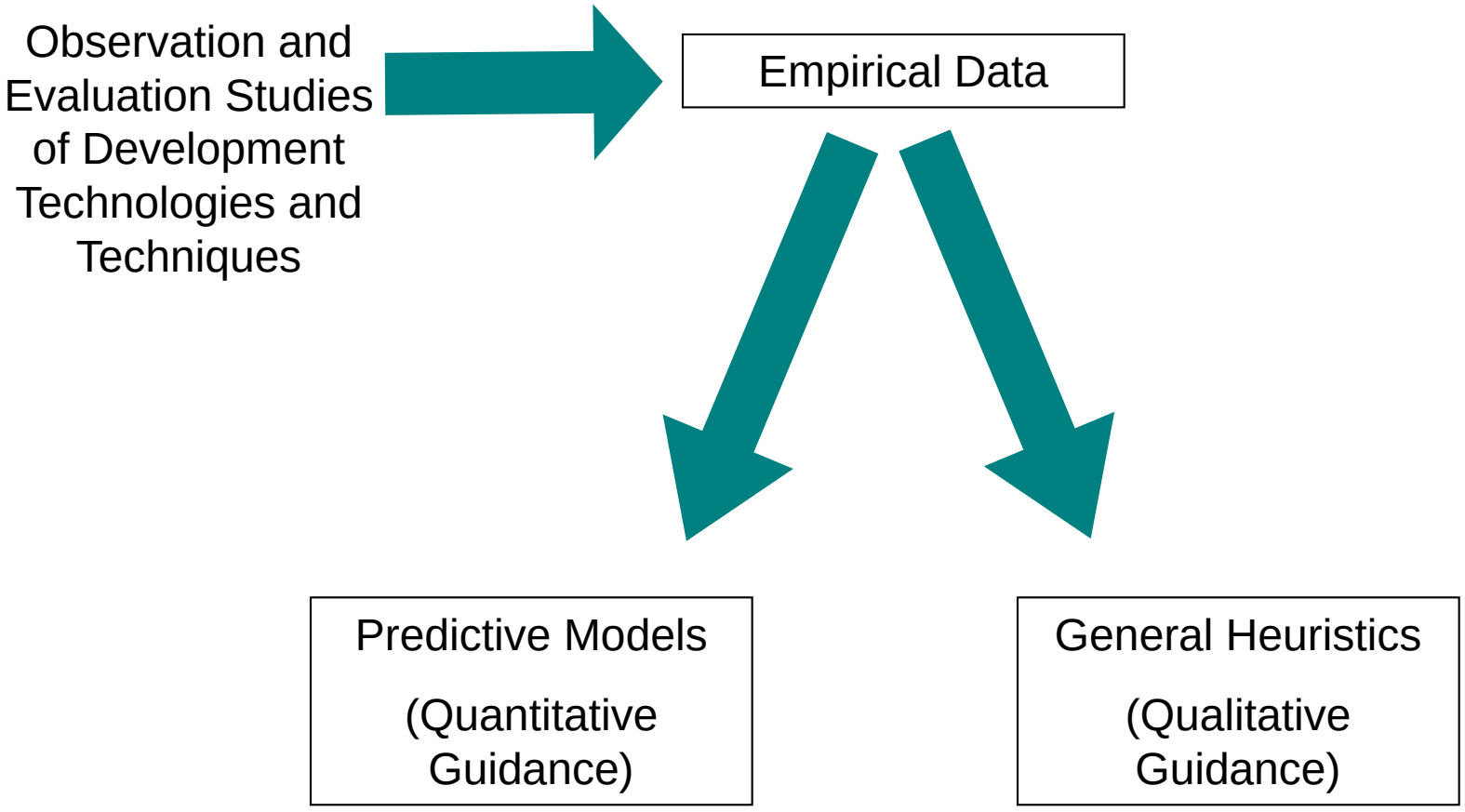
CeBASE Empirical Research Engine Goals

Fraunhofer USA
Center for Experimental Software Engineering

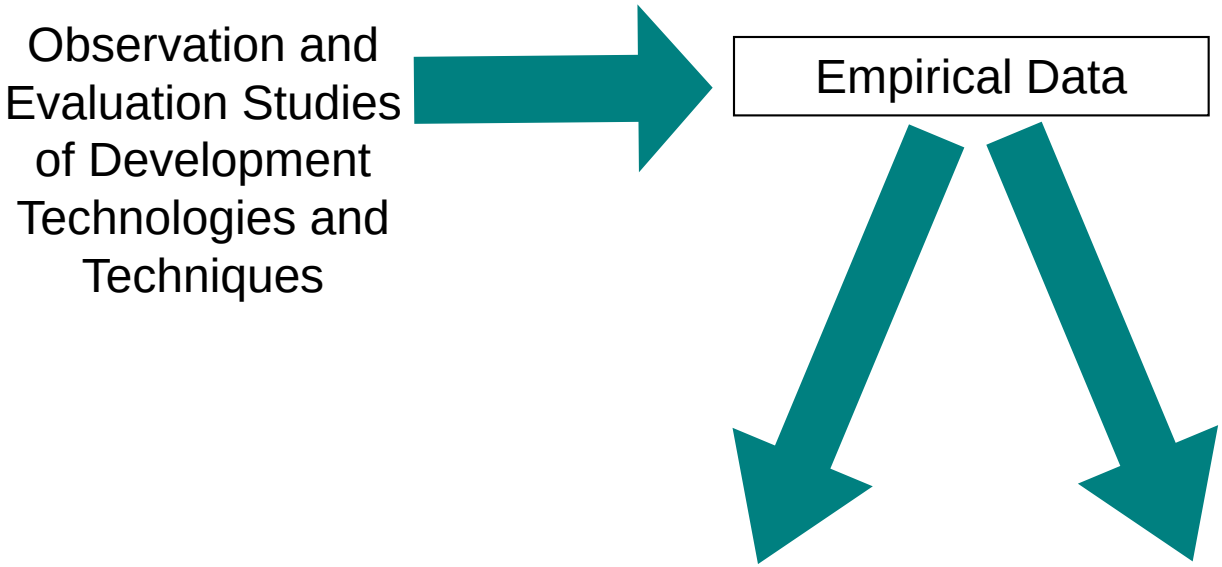
Define and improve methods to

- Formulate evolving hypotheses regarding software development **decisions**
- Collect empirical **data** and experiences
- Record **influencing variables (context)**
- Build **models** (Lessons learned, heuristics/patterns, decision support frameworks, quantitative models and tools)
- Integrate models into a **framework**
- Test **hypotheses** by application
- **Package** what has been learned so far so it can be evolved

CeBASE Decision Framework



CeBASE Experience Base



E.g. COCOTS excerpt:

Cost of COTS tailoring = f(# parameters initialized, complexity of script writing, security/access requirements, …**)**

E.g. Defect Reduction Heuristic:

For faults of **omission** and **incorrect specification, peer reviews** are more effective than functional testing.

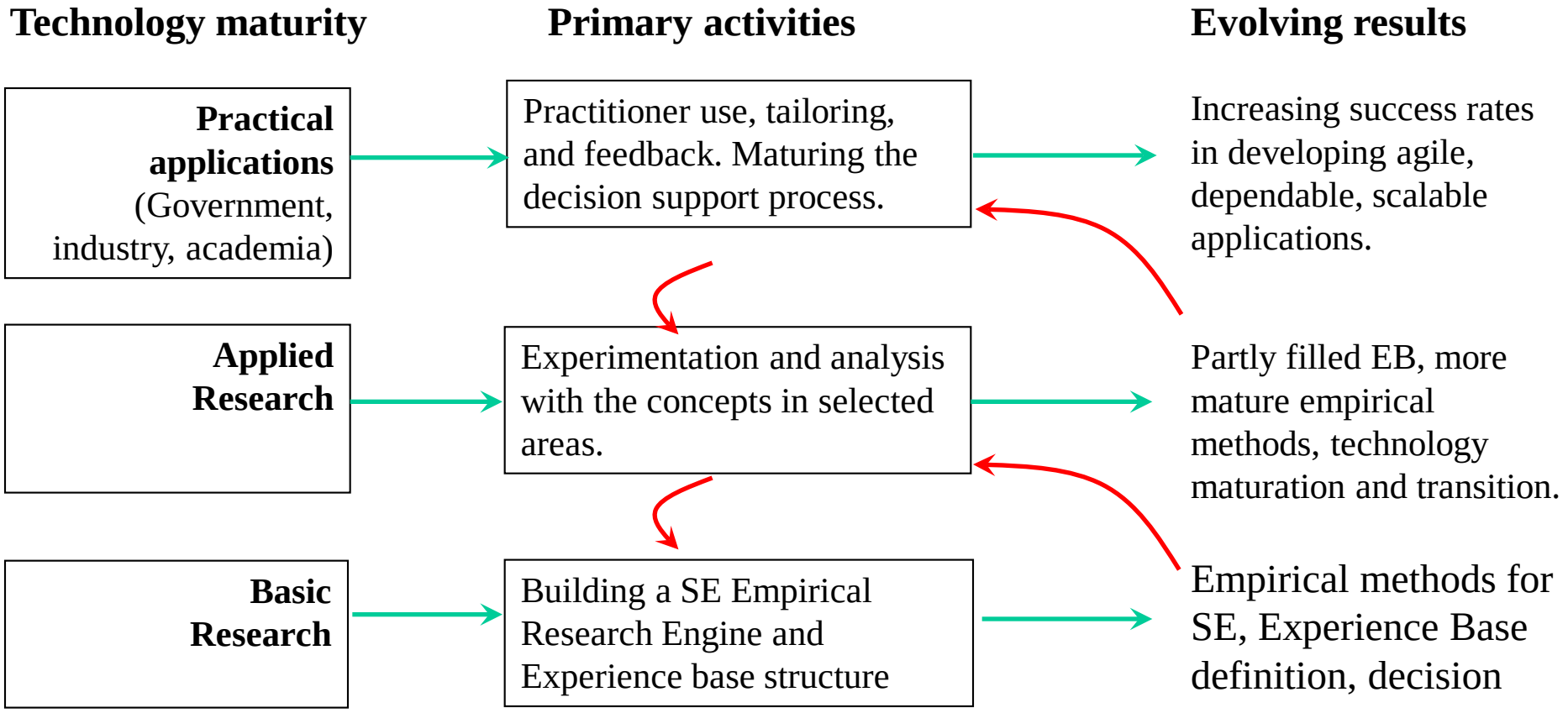
What happened to CeBASE?

Fraunhofer USA
Center for Experimental Software Engineering

- Funding was for 3 years, typically maximum for NSF CS grants
- Recommended that we find other sources of funding
- One source was a large system of systems DoD project in which we were able to continue some of the research but were overcome by the specific project needs
- The second was the NASA software dependability project

CeBASE: Evolving Empirical Evidence Three-Tiered Empirical Research Strategy

Fraunhofer USA
Center for Experimental Software Engineering



Fraunhofer USA
Center for Experimental Software Engineering

Phase IV: Personal Examples

NASA High Dependability Computing Program

Problem: How do you elicit the software dependability needs of various stakeholders and what technologies should be applied to achieve that level of dependability?

Project Goal: Increase the ability of NASA to engineer highly dependable software systems via the development of new technologies in systems like Mars Science Laboratory

Research Goal: Quantitatively define dependability, develop high dependability technologies and assess their effectiveness under varying conditions and transfer them into practice

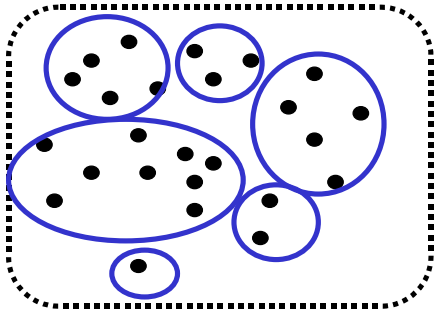
Partners: NASA, CMU, MIT, UMD, USC, U. Washington, Fraunhofer-MD

What are the top level research problems?

Fraunhofer USA
Center for Experimental Software Engineering

System Users

Failures Space

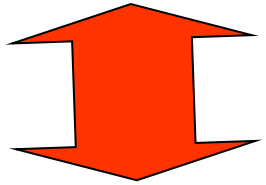


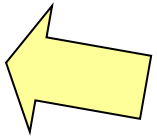
Research Problem 1

How do I elicit quality requirements and express them in a consistent, compatible way?

Research Problem 3

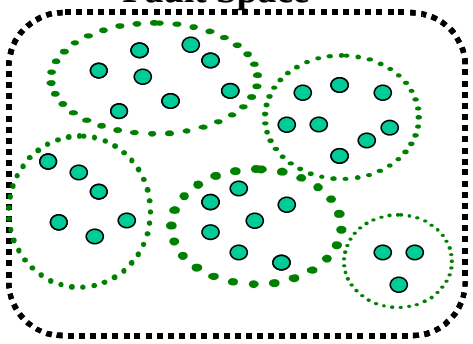
What set of technologies should be applied to achieve the desired quality? (Decision Support)





Technology Developers

Fault Space



System Developers

Research Problem 2

How well does my technology work? Where can it be improved?

UMD - Unified Model of Dependability

Fraunhofer USA
Center for Experimental Software Engineering

- The **Unified Model of Dependability** is a requirements engineering framework for eliciting and modeling quality requirements

- Requirements are expressed by specifying the actual **issue** (failure and/or hazard), or class of issues, that should not affect the system or a specific service (**scope**).

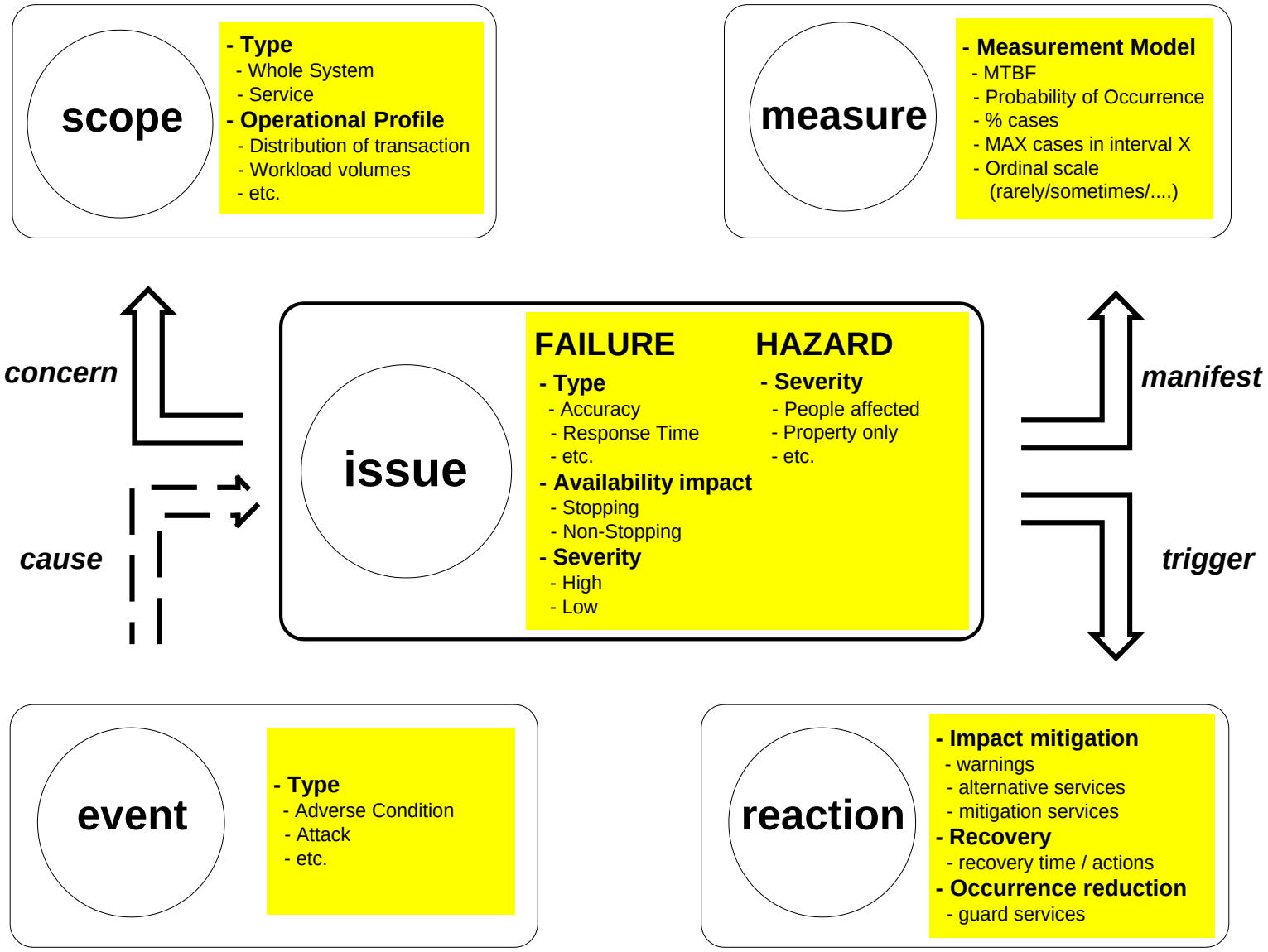
- As issues can happen, tolerable manifestations (**measure**) may be specified with a desired corresponding system **reaction**. External **events** that could be harmful for the system may also be specified.

- For an on-line bookstore system, an example requirement is:

  “The book search service (*scope*) should not have a response time greater than 10 seconds (issue) more often than 1% of the cases (measure); if the failure occurs, the system should warn the user and recover full service in one hour (reaction)”.

UMD is a model builder

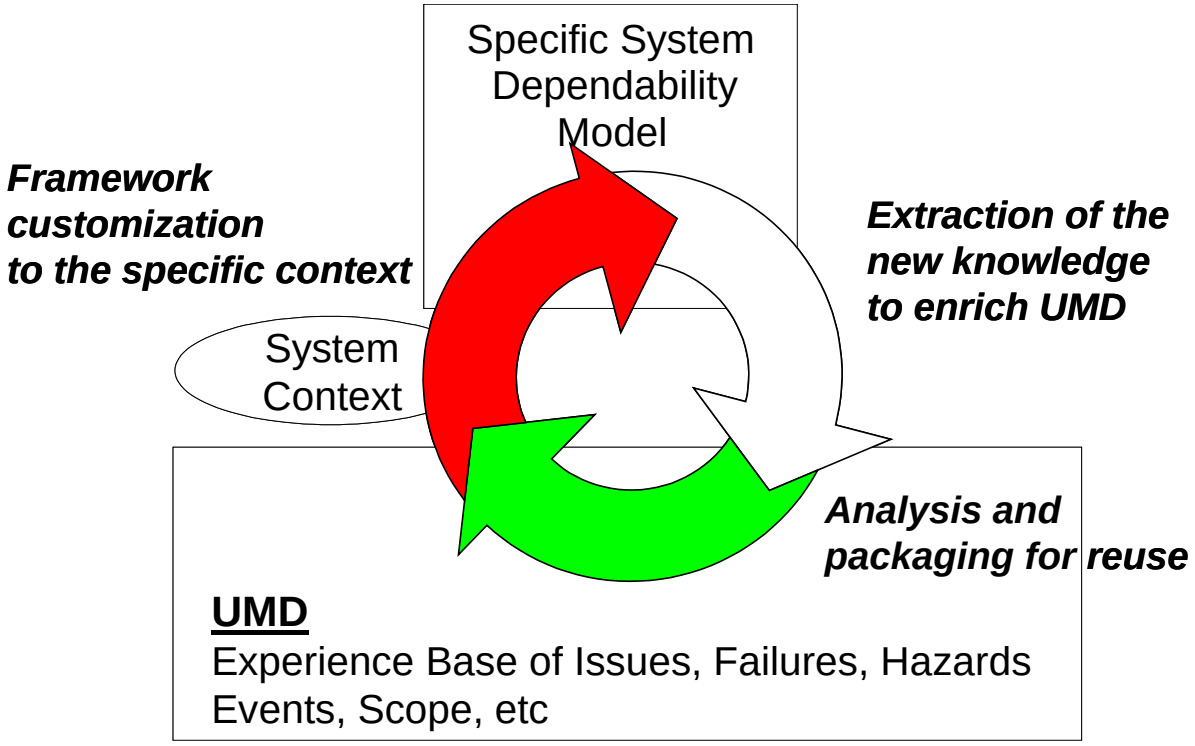
Fraunhofer USA
Center for Experimental Software Engineering



UMD assimilates new experience

Characterizations (e.g., types, severity, etc.) of the basic UMD modeling concepts of issue, scope, measure, and event depend on the specific context (project and stakeholders).

They can be customized while applying UMD to build a quality model of a specific system and enriched with each new application



Using testbeds to transfer technology

Fraunhofer USA
Center for Experimental Software Engineering

- **Define Testbeds**
  - Testbed: Project, operational scenarios, detailed evaluation criteria representative of product needs
  - Can be used to:
    - *Stress the technology* and demonstrate its context of effectiveness
    - Help the researcher *identify the strengths, bounds, and limits* of the particular technology at different levels
    - Provide *insight into the integration* of technologies
    - *Reduce costs* by reusing software artifacts
    - *Reduce risks* by enabling technologies to mature before taking them to live project environments
    - Assist *technology transfer of mature results*

Conduct empirical evaluations of emerging processes

- Establish evaluation support capabilities: instrumentation, seeded defect base; experimentation guidelines

TSAFE testbed: Tactical Separation Assisted Flight Environment

- TSAFE: Aids air-traffic controllers in detecting short-term aircraft conflicts
  - principle component of larger Automated Airspace Computing System by Heinz Erzberger at NASA Ames
  - MIT TSAFE: a partial implementation by Gregory Dennis
    - Trajectory synthesis and Conformance Monitoring
- TSAFE Testbed: developed at FC-MD, based on MIT TSAFE
  - Added testbed specific features, e.g. to monitor faults, output
  - Added features to make it easier to run and experiment with Testbed
  - Synthesized faults that were seeded
  - Added documentation
  - Added functionality, algorithms
- Used to evaluate a family of software architecture techniques
- Used to evolve a particular technique

What happened to HDCP?

- This was meant to be a five year, renewable project in which UMD and USC were subcontractors to CMU
- Two testbeds were developed one at UMD and one at USC
- After 2 year NASA issued a stop work order

Phase IV: Personal Example
DARPA High Productivity Computing Systems

Problem: How do you build sufficient knowledge about the high end computing (HEC) so you can improve the time and cost of developing these codes?

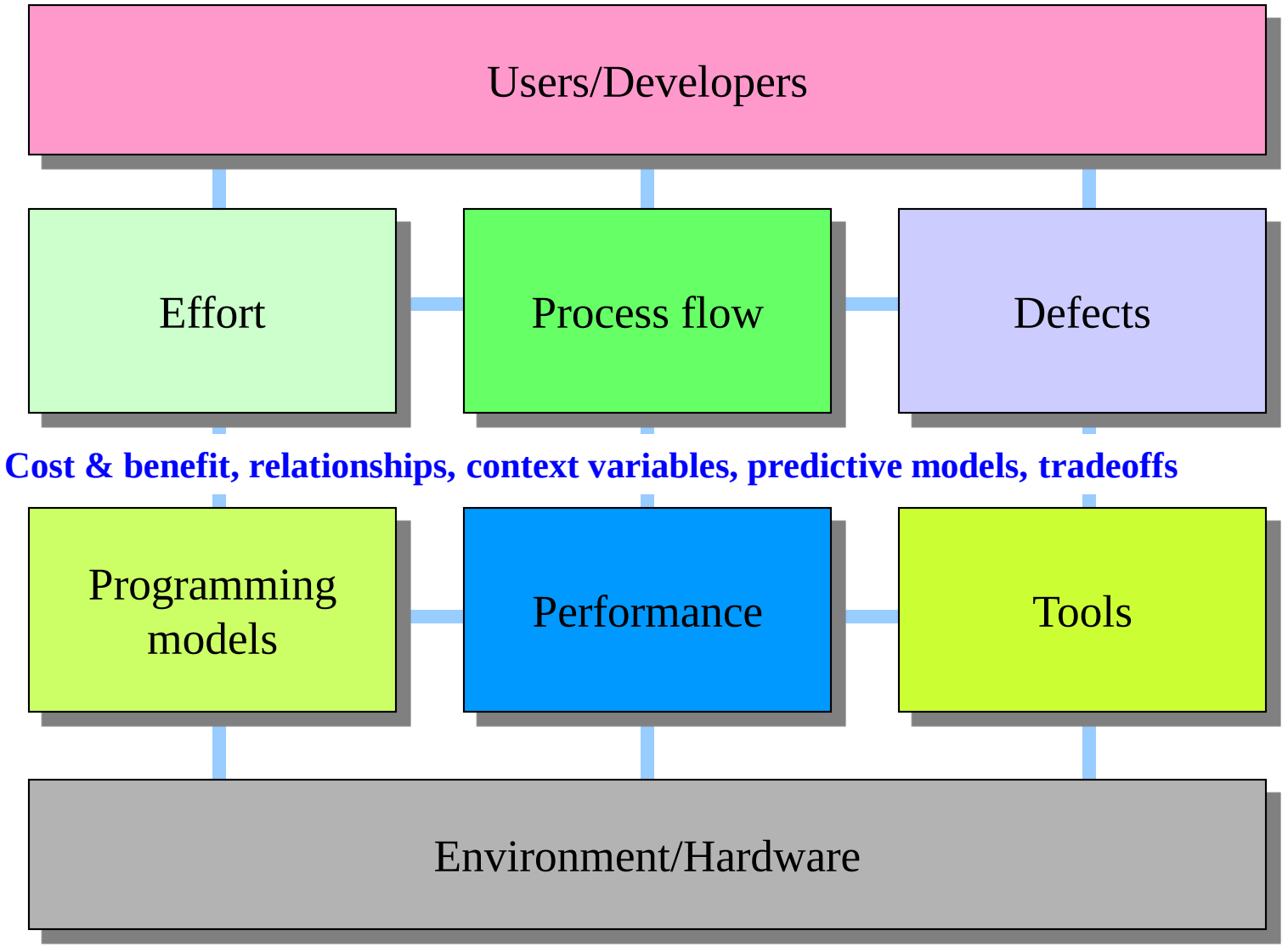
Project Goal: Improve the buyers ability to select the high end computer for the problems to be solved based upon productivity, where productivity means

Time to Solution = Development Time + Execution Time

Research Goal: Develop theories, hypotheses, and guidelines that allow us to characterize, evaluate, predict and improve how an HPC environment (hardware, software, human) affects the development of high end computing codes.

Partners: MIT Lincoln Labs, MIT, UCSD, UCSB, UMD, USC, FC-MD

Areas of Study



Types of Studies

Controlled experiments

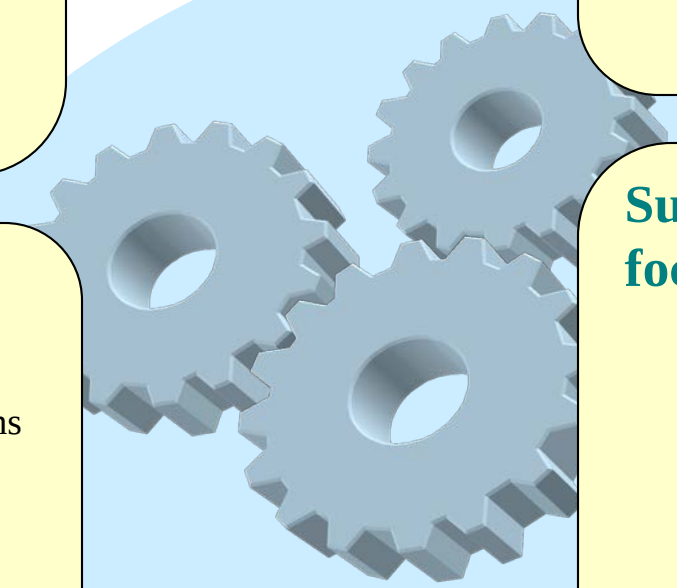
Study programming in the small under controlled conditions to:
Identify key variables, check out methods for data collection, get professors interested in empiricism

E.g., compare effort required to develop code in MPI vs. OpenMP

Observational studies

Characterize in detail a realistic programming problem in realistic conditions to:
validate data collection tools and processes

E.g., build an accurate effort data model



Case and field studies

Study programming in the large under typical conditions

E.g., understand multi-programmer development workflow

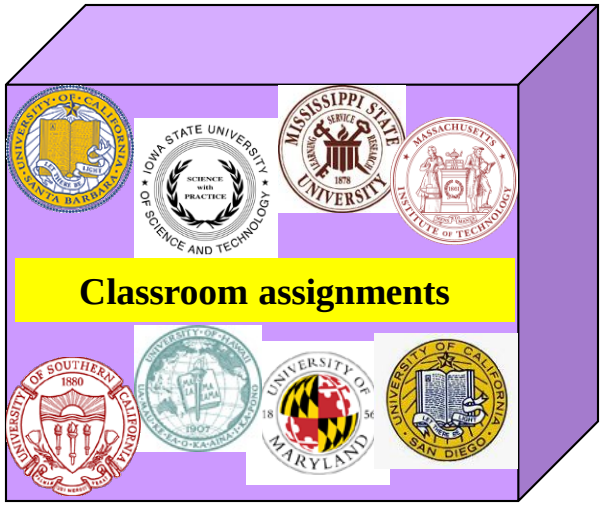
Surveys, interviews & focus groups

Collect “folklore” from practitioners in government, industry and academia

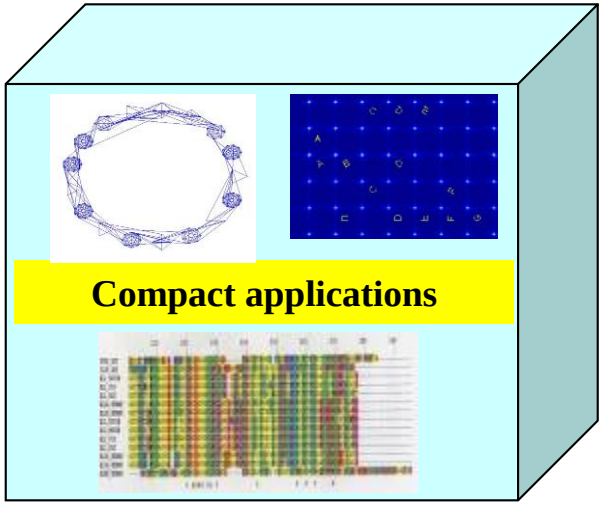
e.g., generate hypotheses to test in experiments and case studies

Types of Testbeds

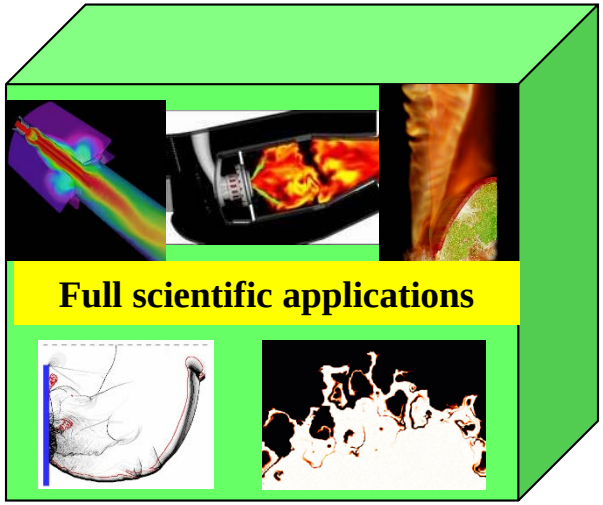
Experimenting with a series of testbeds ranging in size and perspective



Array Compaction, the Game of Life, Parallel Sorting, LU Decomposition,
Developed in graduate **courses** at a variety of universities

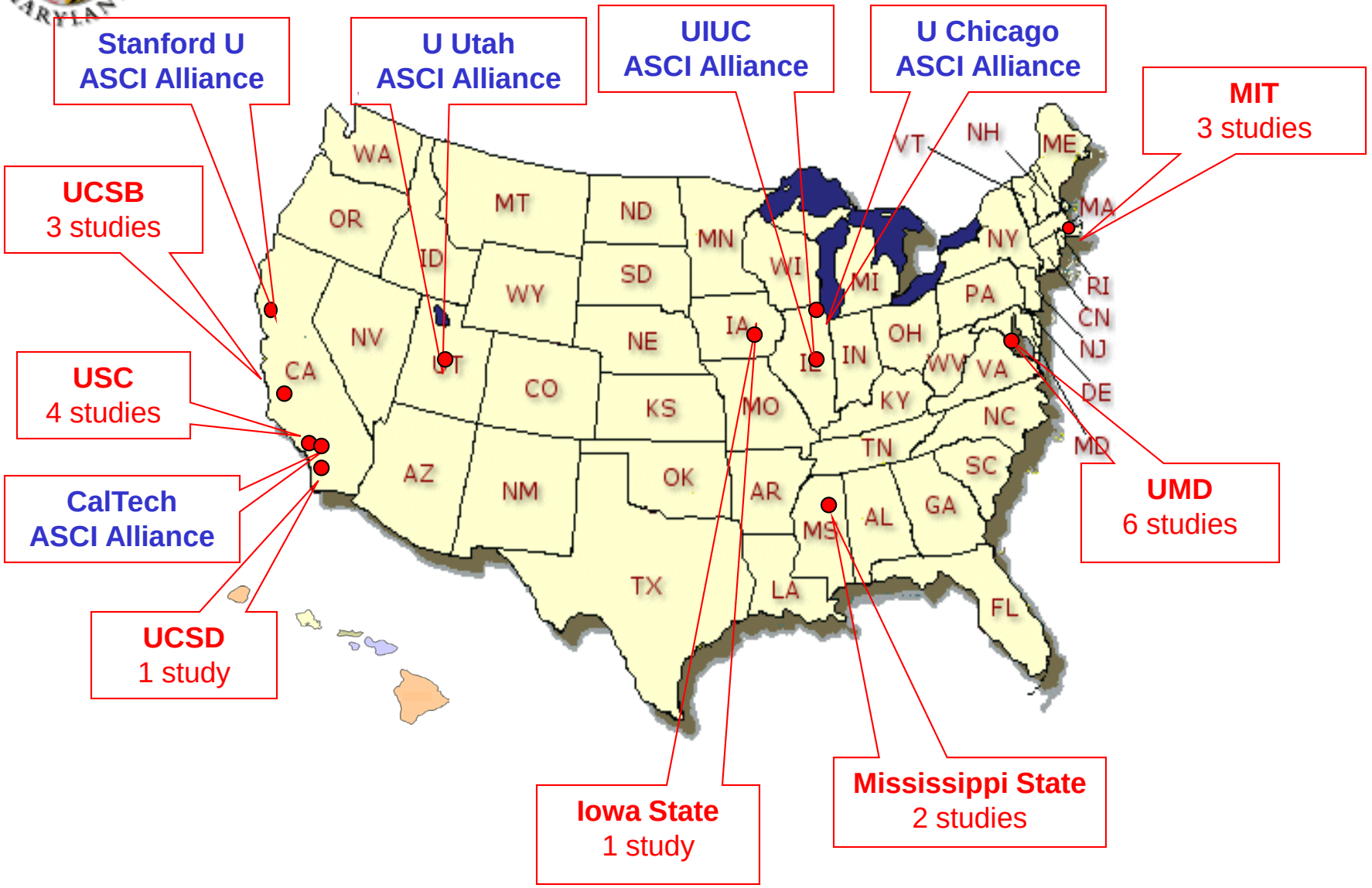


Bioinformatics, graph theory, sensor & I/O: combination of kernels, e.g., Embarrassingly Parallel, Coherence, Broadcast, Nearest Neighbor, Reduction
Developed by experts testing **key benchmarks**



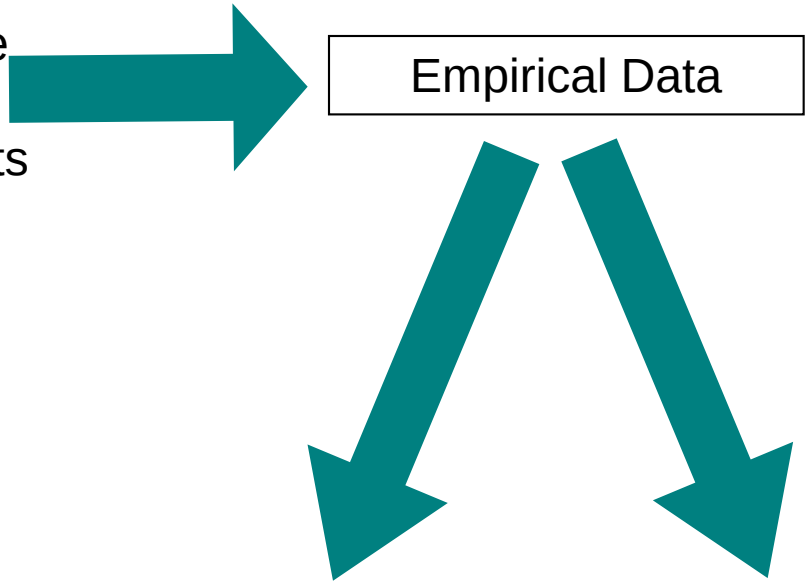
Nuclear simulation, climate modeling, protein folding, …
Developed at **ASCI Centers** at 5 universities
Run at the **San Diego Supercomputer Center**

Studies Conducted



HPCS Experience Base

Development Time Experiments – Novices and Experts



E.g. Tradeoff between effort and performance:

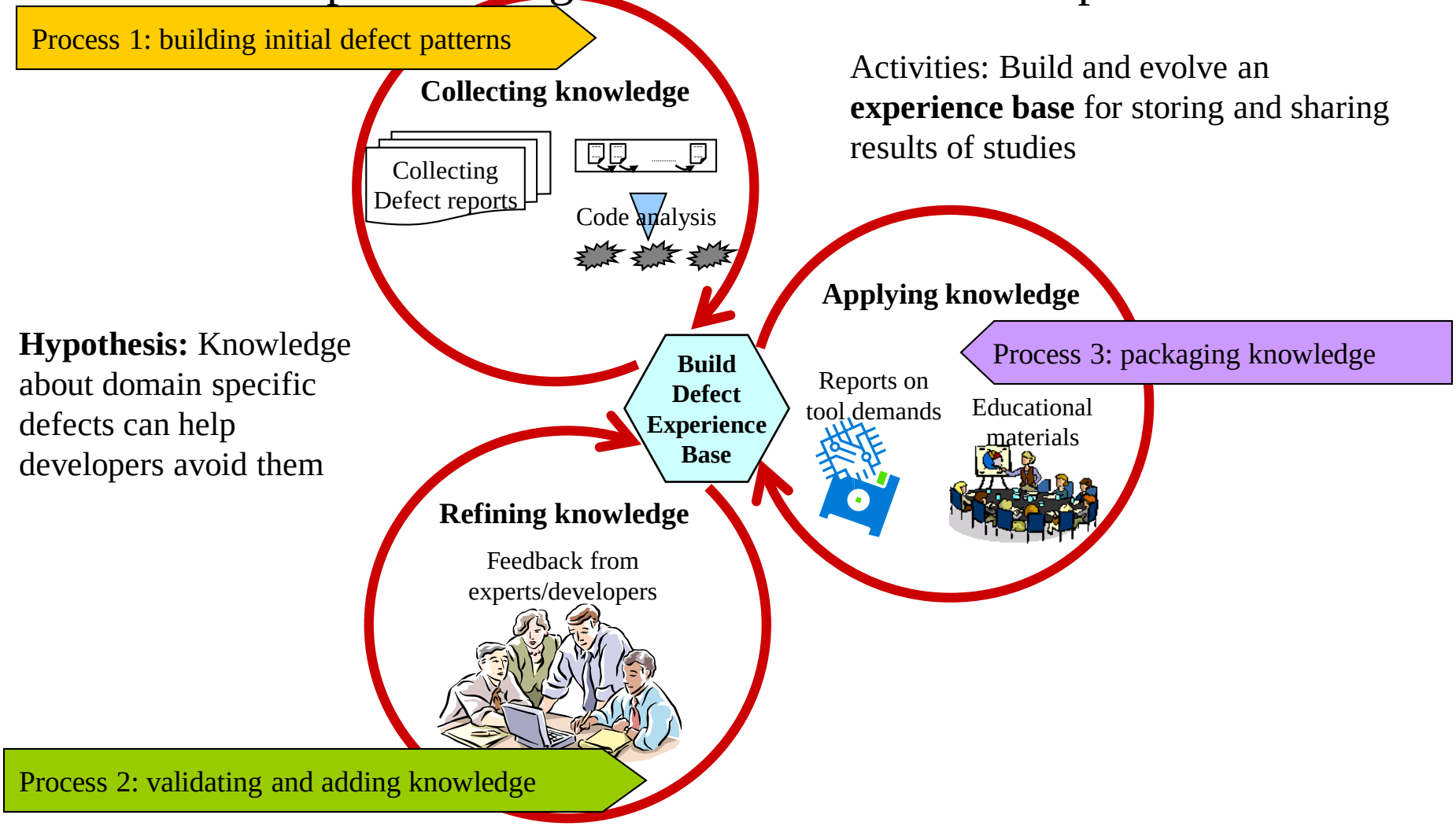
MPI will increase the development effort by y% and increase the performance z% over **OpenMP**

E.g. Experience:

Novices can achieve speed-up in cases X, Y, and Z, but not in cases A, B, C.

Building knowledge about defects

Goal: Provide guidance about types of defects likely to occur during HEC software development using an Iterative/incremental process:



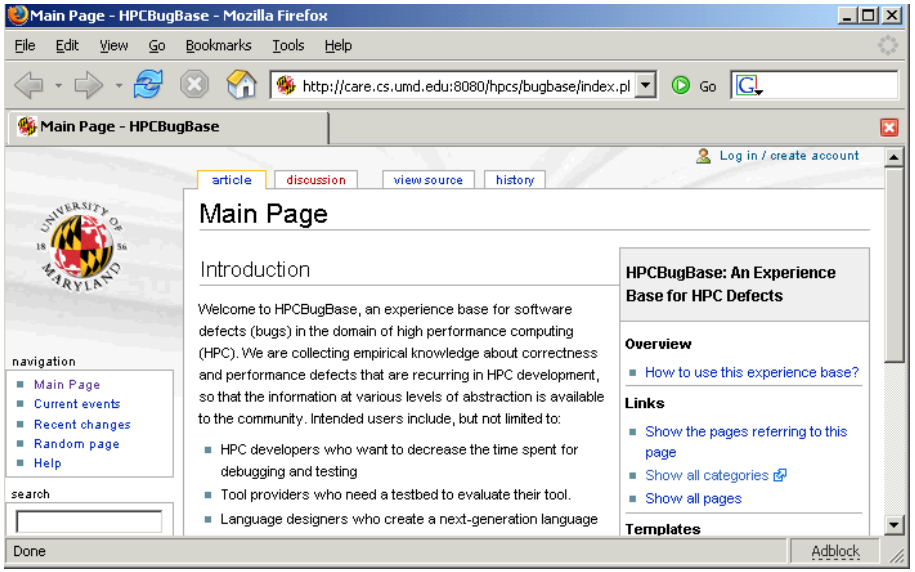
Hypothesis: Knowledge about domain specific defects can help developers avoid them

Building a Defect Experience Base

Fraunhofer USA
Center for Experimental Software Engineering

- We developed a defect experience base for HPC

HPCBugBase (experience base)
http://care.cs.umd.edu:8080/hpcs/bugbase/



Applications

Feedback

- Training materials
- Testbed for tools
- Recommendations to technology providers
- Analysis method

Packaged knowledge

Assist analysis

Data analysis

- Source code history
- Bug tracking systems Mailing lists
- Surveys/interviews

Document recurring correctness/performance problems at various levels of abstraction (source code, defect descriptions, advice, classification schemes)

What happened to HPCS?

Fraunhofer USA
Center for Experimental Software Engineering

- Our role was to provide insights into potential development time improvements, acting as an independent observer and providing those insights to the competitors
- We worked with Cray, IBM and SUN which were down selected from the original five (SGI, HP)
- DARPA decided to cancel our part of the project after 4 years
- Several experience bases essentially lost

Fraunhofer
USA
Center for Experimental Software Engineering

Phase IV

Expanding out across domains, environments, technologies

Lessons Learned:

- *Long term funding* is critical to create an environment for accumulating knowledge
- *Testbeds* are a necessary ingredient for empirical study
- *Domain knowledge* is needed and domain experts need to be part of the experiment team
- *Experience bases* need to be *maintained* over time
- More research is needed to solve the problems of
  - Building a software engineering research engine
  - Building the decision support system with partial knowledge
  - Integrating the process

What I have learned

Fraunhofer USA
Center for Experimental Software Engineering

Experimentation is fundamental to any engineering science

Organizations/Domains have different characteristics, goals, cultures; stakeholders have different needs

Process is a variable and needs to be selected and tailored to solve the problem at hand

We need to learn from our experiences, build software core competencies, build domain knowledge with respect to software

Interaction with various industrial, government and academic organizations is critical to understand the problems

To expand the potential competencies, we must partner

Where are we now?

We have come a long way in evolving the discipline of empirical software engineering

ESE community:

- Many collaborations across international boundaries
- Shared authorship across institutions on many papers
- 23nd ISERN workshop this year

Publications:

- Journal of Empirical Software Engineering (EMSE) in its 20th year
  - 2014 ISI rating higher than IEEE-TSE and ACM-TOSEM
  - 5 year ISI rating higher than IEEE-TSE and ACM-TOSEM
- Most journals now welcome empirical work, in fact many expect it
- ACM/IEEE Empirical Software Engineering and Measurement Symposium (EMSE) in its 13th year
- There are textbooks in the field

But we have a long way to go

Still not enough use of the scientific method, i.e., learning from applying, outside the ESE community (SE and Practice)

Identifying and accounting for context variables is hard

Long term funding is necessary but difficult to acquire and

There is no funding for maintaining experiences bases

Collaborations with ‘practitioner laboratories’ is not easy to do

We need to demonstrate our impact on the practice

What is needed?
Empirical Research Engine

Fraunhofer USA
Center for Experimental Software Engineering

An **experience base** (shared repository) of evolving models and lessons learned that can evolve over time representing what we know about the discipline at any point in time and a collection of **testbeds** for experimentation and evolution of processes

A **decision support system** that provides support for the practice by identifying the benefits, limits, and bounds of techniques, methods, and life cycle models and support for **trade-off decisions**

The real question is: if I want a product to have certain characteristics, e.g., reliability, correctness, safety, security, etc.

what are the appropriate techniques methods, life-cycle models to achieve those characteristics, relative to the context?

What is needed?
Laboratories

Our most important laboratories only exist where software is developed, maintained, etc.

Emphasize the symbiotic relationship between research and practice so both groups can gain and the discipline can evolve

Many laboratories to allow many applications of a process, taking place in different environments, each application providing a better understanding of the concepts and their interaction in context

Over time we need to identify and expand our understanding of context variables

What is needed?
Long Term Support

Our most successful example, the SEL had support for 25 years and so was able to create an environment where long term learning could take place

Shorter term ‘experiments’ can be effective if there is a strong base of shared knowledge and the ability to store and such results in a long term supported experience base

A supported evolving collection of testbeds

What is needed?

Multi-disciplinary teams

Fraunhofer USA

Center for Experimental Software Engineering

Research teams need various forms of expertise, e.g., domain knowledge, software engineering knowledge, a variety of experimentation capabilities

These teams not only provides different levels of expertise but provides checks and balances on the studies themselves

What is needed?
Replications

Fraunhofer USA
Center for Experimental Software Engineering

Replication in software engineering studies is critical but it should be expanding knowledge rather than confirming it

Replication in ESE should play the role of expanding our understanding of the context variables in which existing results may or may not be true

It requires close interaction between the original study team and the replication team, because we cannot always communicate the original context variables

It is hard enough to capture tacit knowledge in replicating experiments, even when the teams are collaborating

What is needed?

Collaborative Communities

Fraunhofer USA
Center for Experimental Software Engineering

Building the tapestry of software engineering knowledge is too big a task for any one group to perform

Empirical software engineering requires groups who share results in effective ways, e.g., via a repository of evolving models and lessons learned that can be used, added to, and evolved by other researchers

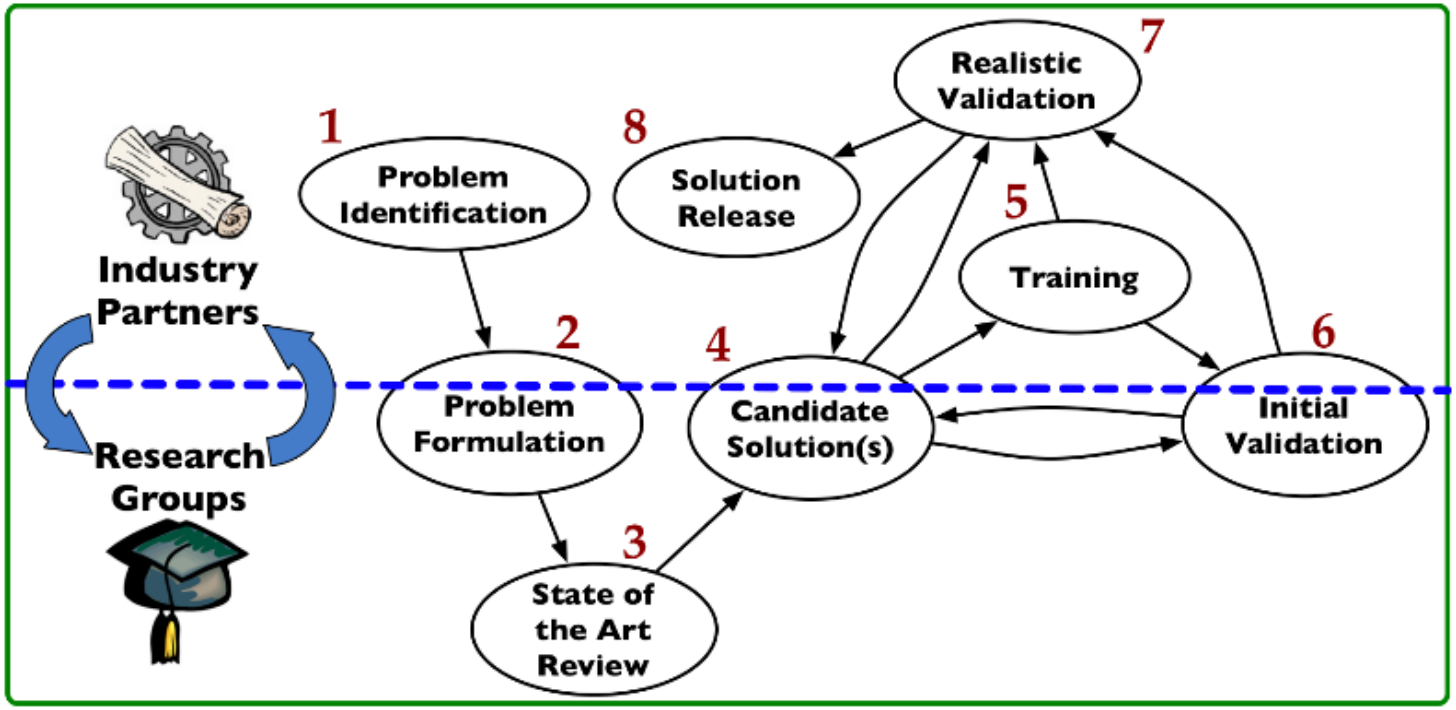
For each group, the focus can be bounded, limiting the context, the domain, the collection of techniques, methods, and life cycle models studied.

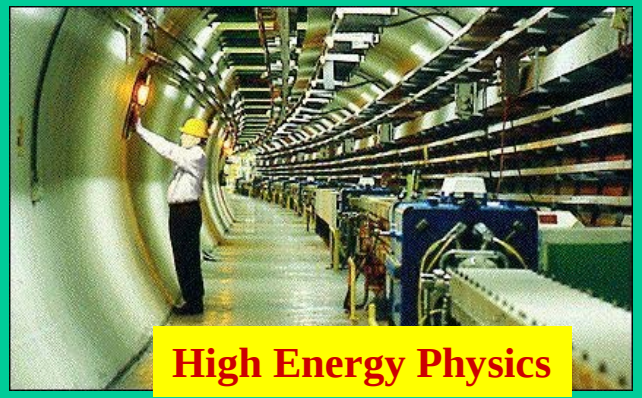
ISERN has been very successful but it is not enough

Software Validation and Verification Lab
University of Luxembourg

Mode of Collaboration

- Strong emphasis on applied research, driven by needs
- Tight, large-scale industrial collaborations

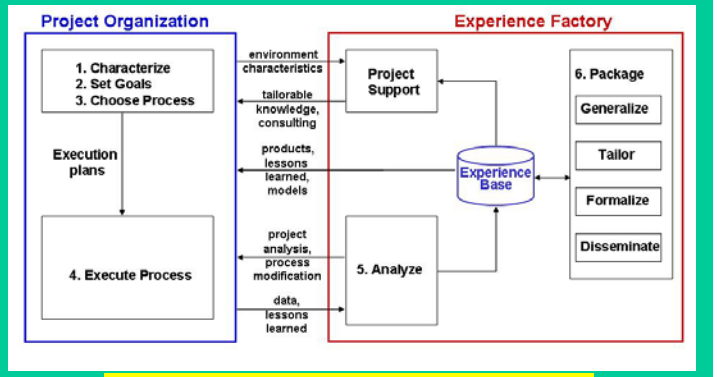




High Energy Physics

Space Science

High Performance Computing



Software Engineering

Software Engineering is “**big science**”;
not small independent technology developments

Contributing Team

Rola Alameh, Sima Asgari, John W. Bailey, Vic Basili, John Beane, Barry Boehm, Lionel Briand, Carolyn Brophy, Gianluigi Caldiera, Giovanni Cantone, Jeff Carver, Steve Condon, Patricia Costa, Daniela Soares Cruzes, M.K. Daskalantonakis, Alex Delis, Carl Doerflinger, Paolo Donzelli, Sandra Fabbri, Karl Freburger, Scott Green, Loren Hochstein, Dave Hutchens, Bok Gyu Joo, Natalia Juristo, Beth Katz, Yong-Mi Kim, Jyrki Kontio, Ara Kouchakdjian, Oliver Laitenberger, Filippo Lanubile, Lucas Layman, Mikael Lindval, Chris Lott, Jose C. Maldonado, Vladimir Mandic, Maurisio Morisio, Yasuhiro Mashiko, Frank McGarry, John McHugh, Walceilo Melo, Manoel Mendonca, Sandro Morasca, Taiga Nakamura, Markku Oivo, Jerry Paige, Rose Pajerski, Nikki Panlilio-Yap, Barry Perricone, Tsai-Yun Philips, Connie Loggia Ramsey, Jim Ramsey, Robert Reiter, Dieter Rombach, Ioana Rus, Alessandro Sarcia, Carolyn Seaman, Rick Selby, Forrest Shull, William Thomas, Koji Torii, Guilherme Horta Travassos, Joe Turner, Roseanne Tesoriero Tvedt, Jon D. Valett, Sira Vegas, Sharon Walagora, Dave Weiss, Daniil Yakimovich, Nico Zazworka, Marv Zelkowitz, Zhijun Zhang, ….

http://www.cs.umd.edu/~basili/papers.html

Can we change the culture?

- A community built experience bases/repositories, added to and supported by all parties
- Maybe start by gathering topic areas supported by the relevant researchers and users
- Financed by government, a community of companies, … (Hubble Telescope)
- Rewards for academics and practitioners include
  - Academic papers with multiple authors, e.g., the theory builders, prior experimenters, tool builders, knowledge maintainers, experience (physics papers)
  - Credit/acknowledgement for adding to and using the EB, (open source)
  - …

- Is this possible? I don't know